Department of Computer Science

Technical Report

MSc Dissertation: Tuning of rsync algorithm for optimum cloud storage performance

Maria Constantinou

Technical Report 2013-10 **December 2013**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

Tuning of rsync algorithm for optimum cloud storage performance

Maria Constantinou
MSc in Modern Applications of Mathematics

2013

Tuning of rsync algorithm for optimum cloud storage performance

Submitted by Maria Constantinou
for the Degree of MSc
of the University of Bath

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Mathematical Sciences. No portion of the work in this thesis has been submitted in support of an application for any other degree or qualication of this or any other university or institution of learning. Except where specically acknowledged, it is the work of the author.

Abstract

The rsync algorithm is tuned so to obtain optimum cloud storage performace. More than 50000 emulations with the "real rsync" and real files have been carried out so to obtain information on the algorithm's performance. The performance of cloud storage is said to be optimum when the number of bytes trasmitted with every synchronisation is minimum. Hence I also present a model that captures the number of bytes that are transmitted through the rsync link. This model was validated by simulating it and comparing the results to the outcomes of the experiments that were performed. Finally by analysing the model mathematically I extracted an expression that gives the optimal block-size which depends on the size of the file and the changes made to it.

Acknowledgements

I am thankful to my two project supervisors Professor James Davenport at the University of Bath and Dr Keith Briggs at BT Research for their patience, guidance and support throughout this dissertation. Let me be more specific and say a tremendous "Thank you" for their immediate responses to all my hopeless emails.

Special mention must also go to Professor Chris Budd for always smiling and being so positive about my mathematical abilities and strenghts.

I am grateful to my parents for making me a patient hard worker and who planted the desire for knowledge and determination to achieve my goals.

Last but not least I am beholden to all the friends (or even more than friends) who were next to me during the hard research period.

Contents

List of Figures

1 Introduction

1.1 Cloud online storage

The project's subject is Cloud online storage and has been carried out in cooperation with British Telecommunications. In our days Cloud storage is widely used. BT Broadband at home has the Cloud online storage feature.

BT Cloud, being an auto backup service, keeps a copy of everything on the users' devices, so the most recent versions of their files are always protected. Moreover online storage helps in freeing up space on their devices. Using BT Cloud app the user can access files and photos wherever they are and on the move. Furthermore BT Cloud enables the user to share files with friends and family via email, Facebok and Twitter[1].

Now, let me give an idea on how this operates by giving a classic example from a user's daily routine. Suppose the user works from home on a document which is stored on the Cloud and then they turn off their computer to go to work. While they were working at home once they made a change on the document, the version of the document on the Cloud was automatically synchronised. Moreover, suppose that the last time the user accessed this specific document from work was about 36 hours ago. Obviously when the user gets back to work they want to continue working on this document from the point they were left at home. This is what Cloud Storage does for them. As soon as they get to work they can download the updated document from the Cloud and continue working on the version that was synchronised while they were working from home.

Figure 1: Cloud storage [2]

1.2 The rsync algorithm

BT Cloud backup is achieved using the rsync algorithm given in Andrew Tridgell's PhD thesis "Efficient Algorithms for Sorting and Synchonization" [3]. The thesis also describes several applications of the remote data update algorithm (rsync algorithm) and uses of the ideas behind it. Incremental backup, which is actually the style of backup used by the Cloud, is one application of the rsync algorithm. In simple words, incremental backup starts with a full backup. Then all subsequent backups have to do with only the blocks that have changed since the first full backup [4].

Here, I describe how the rsync algorithm operates.

Suppose there are two file systems A and B connected by a low-bandwidth high-latency link. Say, also, that B is the remote file system and A the local file system. This means that the algorithm aims to send a copy of the file from A to B. Before the synchronisation A has a file with bytes a_i and B has a file with bytes b_i ($0 \leq i < n$ i.e. each file is n bytes long). This is illustrated in Figure 16.

The rsync algorithm performs synchronisation by exchanging block signature information. Two signatures are being used. The first signature is fast and cheap and it is computed at every byte offset in a file. The second signature is strong and expensive and it has a very low probability of random collision. This signature is computed by A only at byte offsets where the fast signature matches one of the fast signatures from B. Denote the fast signature R and the strong one H.

The structure of the rsync algorithm is the following:

1. B divides b_i into N equally sized blocks b'_j and computes signatures R_j and H_j on each block. These signatures are sent to A.
2. For each byte offset i in a_i, A computes R'_i on the block starting at i.
3. A compares R'_i to each R_j received from B.
4. For each j where R'_i matches R_j, A computes H'_i and compares it to H_j.
5. If H'_i matches H_j then A sends a token to B indicating a block match and which block matches. Otherwise A sends a literal byte to B.
6. B uses the literal bytes and tokens, received from A, to construct a_i.

Let me note that each comparison is done at each byte boundary of the local file in A and at each block boundary of the remote file in B so the algorithm is able to find matches at non-block offsets. This allows for arbitrary length insertions and deletions between the local and remote files to be handled.

These comparisons are done very efficiently by a hash search algorithm. More information about hash functions can be found in the Handbook of Applied Cryptography by A. Menezes, P. van Oorschot, and S. Vanstone [5]. The received signatures for each block of the remote file are stored in a table and indexed by a 16 bit hash. This is illustrated in Figure 2. Then when the fast signature at each byte offset of the local file is computed is stored and indexed by a 16 bit hash as well. A linear search is then performed through the two lists, stopping when an entry is found with a 16 bit hash which doesn't match. For each entry the 4 byte fast signature is compared to the entry of the signature table. If that matches the full 16 bytes strong signature

is computed at the current offset and compared to the strong signature in the signature table. This basically explains what happens inside the steps 3 and 4 of the rsync algorithm.

In simple words the algorithm performs a filtering aiming to discover where exactly the changed file is different from the existing so to transmit just the changed parts of it.

1.3 Problem description

The synchronisation link experiences a huge amount of network traffic, as well as latency, while a remote copy of a file is being synchronised with the changed local one.

As mentioned before, rsync algorithm is used for synchronising remote copies of files to local ones. My main aim is to optimize the performance of the rsync algorithm given in Andrew Tridgell's PhD thesis [3]. Here "optimise" means to minimise the amount of network traffic generated when small changes are made to large files.

1.4 Main steps

Firstly, I was provided with some statistics on file-size distribution from two sources.

I continued by getting some statistics on the performance of the standard version of rsync, by performing emulations. Python scripts were used that call the "real rsync" algorithm. This way I ran a large number of data transfer. The performance metric was the ratio of the size of the changed file to the number of bytes actually transferred. This is defined as speedup in Andrew Tridgell's thesis [3].

From this thesis, I extracted a minimal description of the algorithm so to identify those adjustable parameters which control its performance. Being motivated by an example mathematical model that is given in the thesis [3] and synthesising the outputs from above, I constructed a mathematical model for the number of bytes trasmitted with every synchronisation.

After validating the model I analysed it mathematically. This way I adjusted the tunable parameters so to optimize the performance of the algorithm.

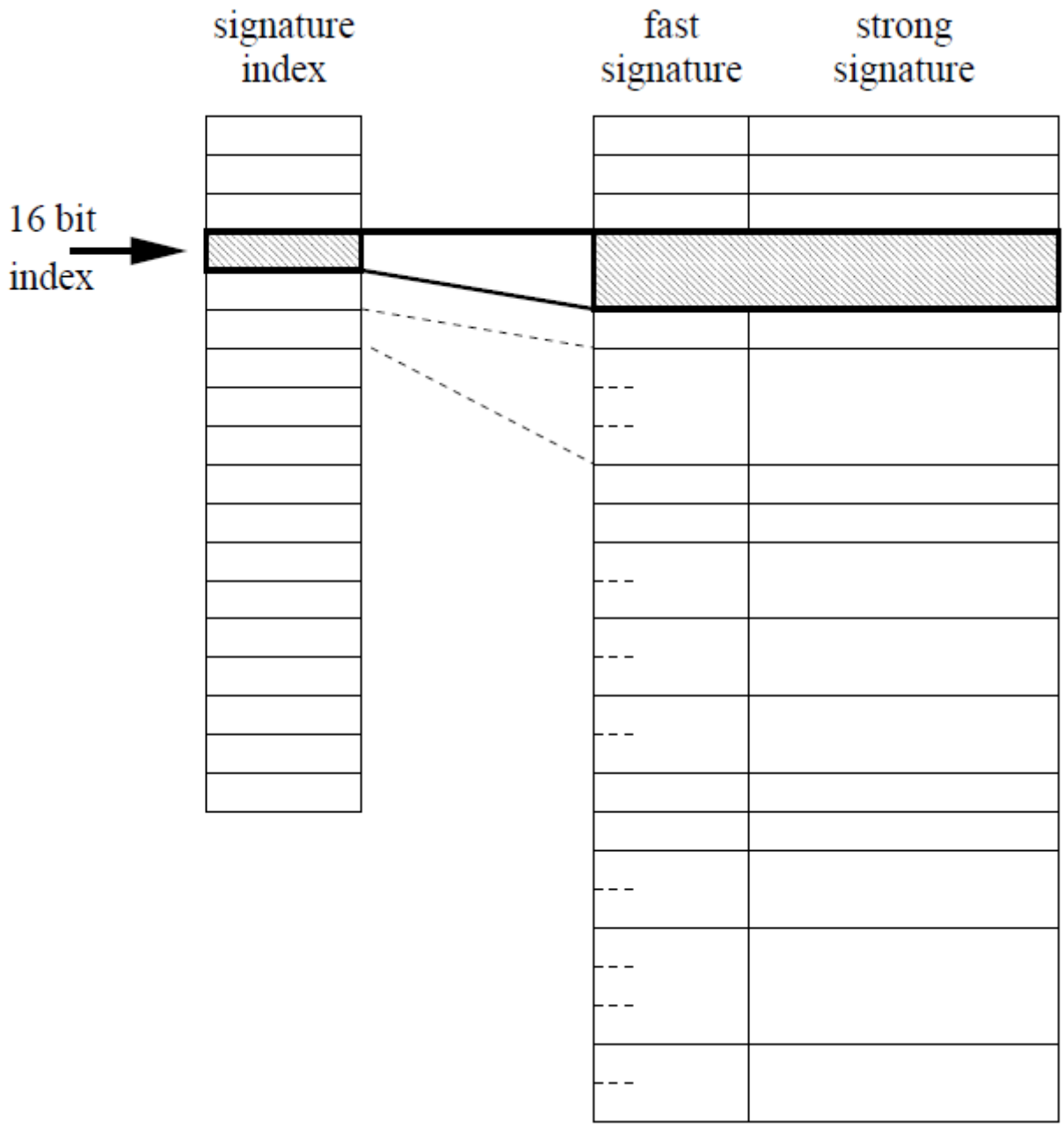

Figure 2: The index table and the sorted signatures table of the signature search algorithm [3]

2 The file-size distribution

It is assumed that the file-size distribution is power-law. Initial information about the file-size distribution has been found in the paper on "File size distribution on UNIX systems-Then and now" by Andrew S. Tanenbaum [7]. Then the power-law distribution assumption was made considering the Figures 3 and 4 provided by J. Davenport, University of Bath and K. Briggs, BT respectively. Figure 3 illustrates the file-size distribution of a real file system. It can be said that this indicates that the file-sizes follow the power-law distribution. Figure 4 illustrates a $\log - \log$ plot of the cumulative distribution function of the file-sizes in the real file system. The approximate straight line indicates a power-law as well.

Furthermore typical parameters for the power-law distribution had to be estimated. They were extracted from the plot in Figure 4 provided by K. Briggs, BT.

This plot can be approximated by a straight line so the equation that corresponds to it is

$$\log(y) = \text{constant} + n\log(x). \tag{2.1}$$

The parameter n was calculated. It is actually the power parameter of the distribution.

$$n = \frac{10^{0.25} - 10^{-4.5}}{1 - 14} = -0.137$$

Hence, the power-law distribution is as follows:

$$P(x) = Cx^n \text{ for } x \in [x_0, x_1] \Longrightarrow P(x) = Cx^{-0.137} \text{ for } x \in [10^1, 10^{14}]. \tag{2.2}$$

The distribution, $P(x)$, is used in simulations and emulations in Python, so for coding reasons I had to generate it using a Uniform distribution.

Assume that z is a value of the random variable Z which is uniformly distributed on $[0, 1]$. In E. W. Weisstein [8] it is discussed that the random variable

$$X = [(x_1^{n+1} - x_0^{n+1})z + x_0^{n+1}]^{\frac{1}{n+1}} = [(1.208 \times 10^{12})z + 10^{0.863}]^{\frac{1}{0.863}}$$

is distributed as

$$P(x) = Cx^{-0.137} \text{ for } x \in [10^1, 10^{14}] \text{ and } C = \frac{n+1}{10^{14\times(n+1)} - 10^{(n+1)}} = 7.15 \times 10^{-13}. \tag{2.3}$$

Hence, I used the above argument to generate random numbers that come from a power-law distribution. As we will see in the following sections, in my codes, the range $[10, 10^{14}]$ was replaced by the range $[e, e^{14}]$ so to deal with smaller files and make simulations and emulations faster. This means that $[x_0, x_1] = [e, e^{14}]$.

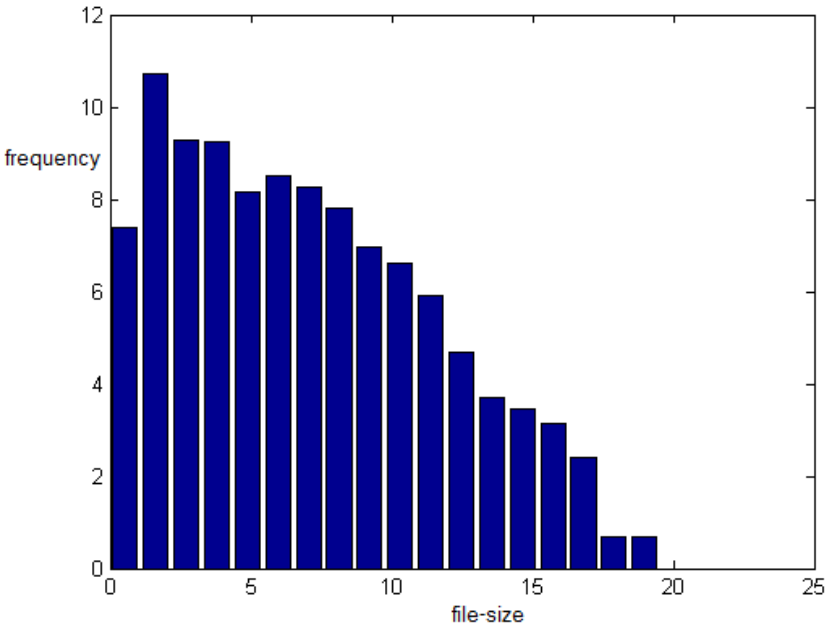

Figure 3: File-size distribution of file system 1 (provided by J. Davenport, University of Bath)

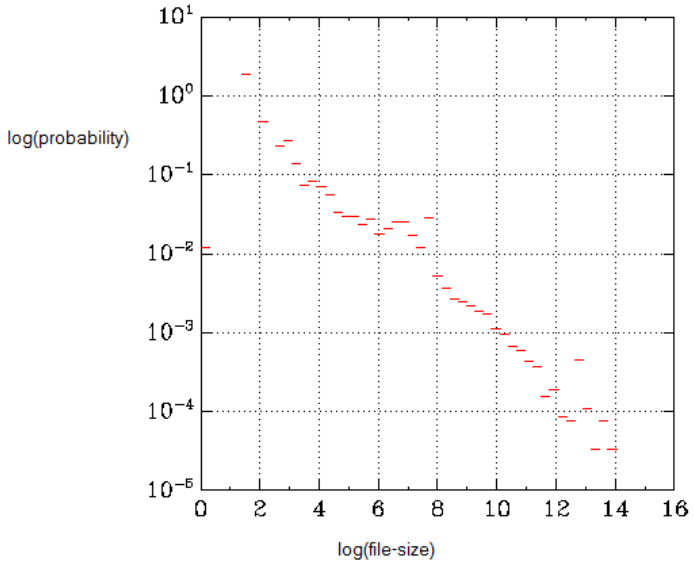

Figure 4: File-size distribution of file system 2 (provided by K. Briggs, BT)

3 My experiments

My experiments were performed using the "real rsync" and real files. They were coded in Python. The codes for these experiments are in Appendix E.

I work with one file that contains a random string of bytes. The size of the string is randomly chosen from the power-law distribution presented in Section 2. This gives an average file-size of 546KB. First I create such a file and I copy it, then picking a random byte position I delete c bytes and add r bytes so a modified version of the copied file is created. Both c and r come from a power-law distribution that is steeper than the one the file-size follows. The average value of c and r is 18 bytes and 28 bytes respectively. Then synchronisation is performed by accessing the modified file remotely.

Here is a list of the initial experiments carried out:

1. The two files were synchronised setting the block-size parameter in the rsync algorithm equal to 1KB. I performed this emulation 10000 times. The results are illustrated in Figure 5.

2. The two files were synchronised setting the block-size parameter in the rsync algorithm equal to 2KB. I performed this emulation 10000 times.The results are illustrated in Figure 6.

3. The two files were synchronised setting the block-size parameter in the rsync algorithm equal to 4KB. I performed this emulation 10000 times. The results are illustrated in Figure 7.

4. The two files were synchronised setting the block-size parameter in the rsync algorithm equal to 6KB. I performed this emulation 10000 times. The results are illustrated in Figure 8.

5. The two files were synchronised setting the block-size parameter in the rsync algorithm equal to 8KB. I performed this emulation 10000 times. The results are illustrated in Figure 9.

In Figures 5, 6, 7, 8 and 9 we can see how the speedup distribution behaves for different values of the block-size parameter. Let me highlight that the highest mean for the speedup is observed when the block-size value is 2KB.

Figure 10 was produced by J. Davenport using the codes in Appendix E. It illustrates the relationship of the file size and speedup when a single change is made to the local file and the block-size parameter in rsync is fixed. This is almost linear. The obvious thing here is that when the file size increases, the speedup increases. This leads us to the conclusion that the sparser the changes the larger the achieved speedup.

In addition performing 1000 emulations each time I obtained the Figures 11, 12 and 13. They illustrate the speedup distribution for block-size value 2KB. 20, 50 and 100 1-byte changes were made to the local file respectively and then the original version of the file was synchronised with the changed one. The speedup values observed in Figures 11, 12 and 13 are much lower than the values observed in the case of a single change (Figure 6). There are, even, some speedup values less than 1. This means that the bytes transmitted through the rsync link were more than the size of the changed file.

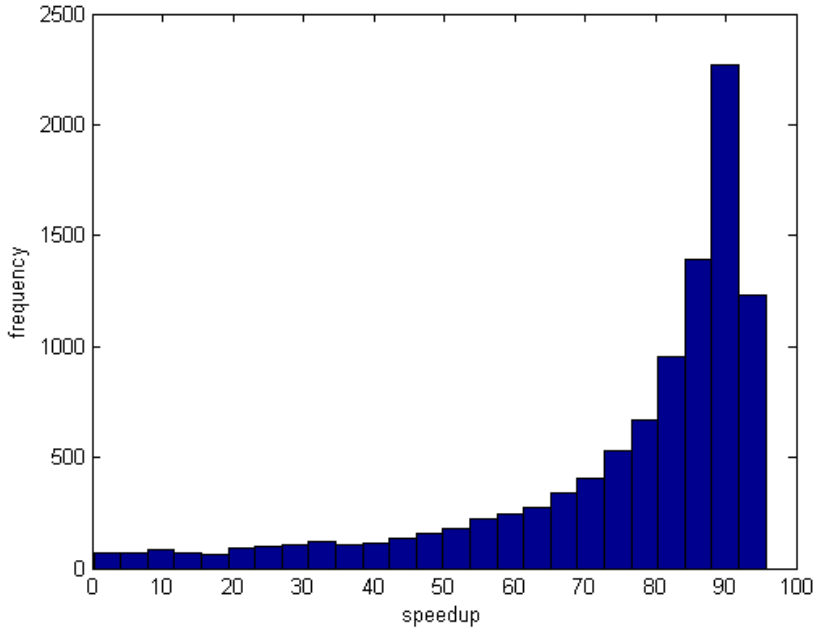

Figure 5: Distribution of the speedup of the rsync algorithm with the block-size parameter set to 1KB

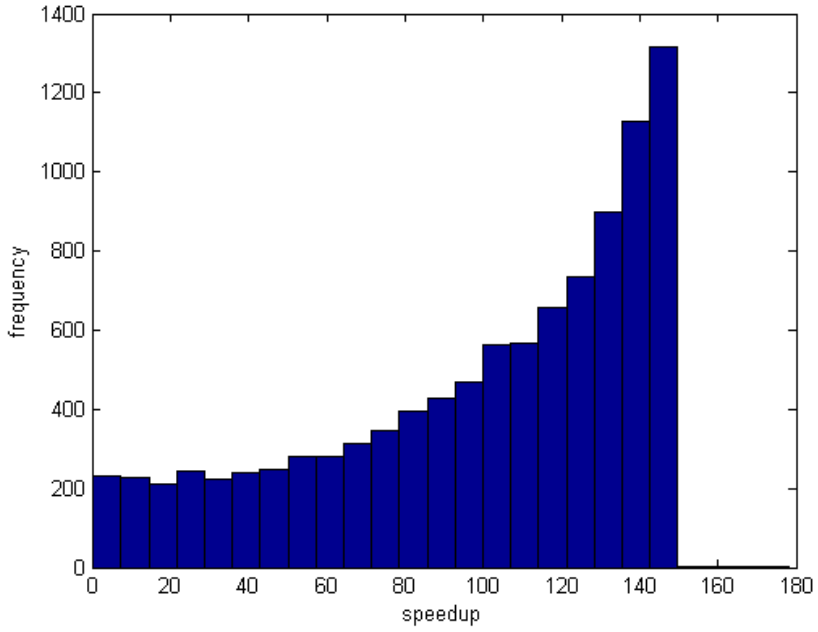

Figure 6: Distribution of the speedup of the rsync algorithm with the block-size parameter set to 2KB

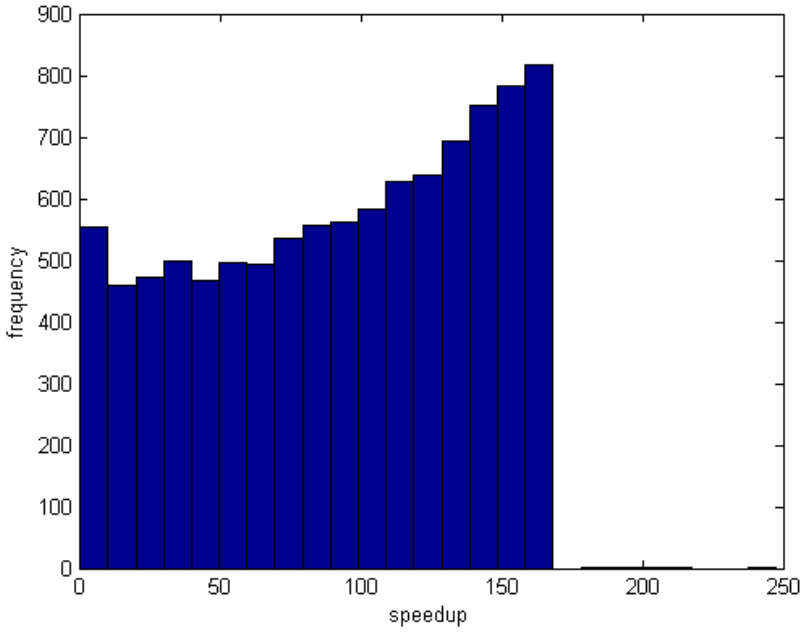

Figure 7: Distribution of the speedup of the rsync algorithm with the block-size parameter set to 4KB

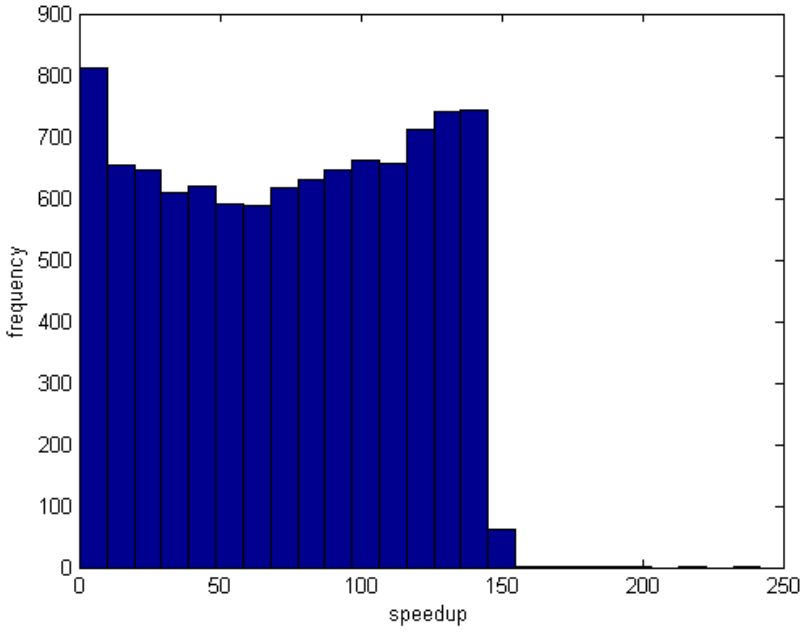

Figure 8: Distribution of the speedup of the rsync algorithm with the block-size parameter set to 6KB

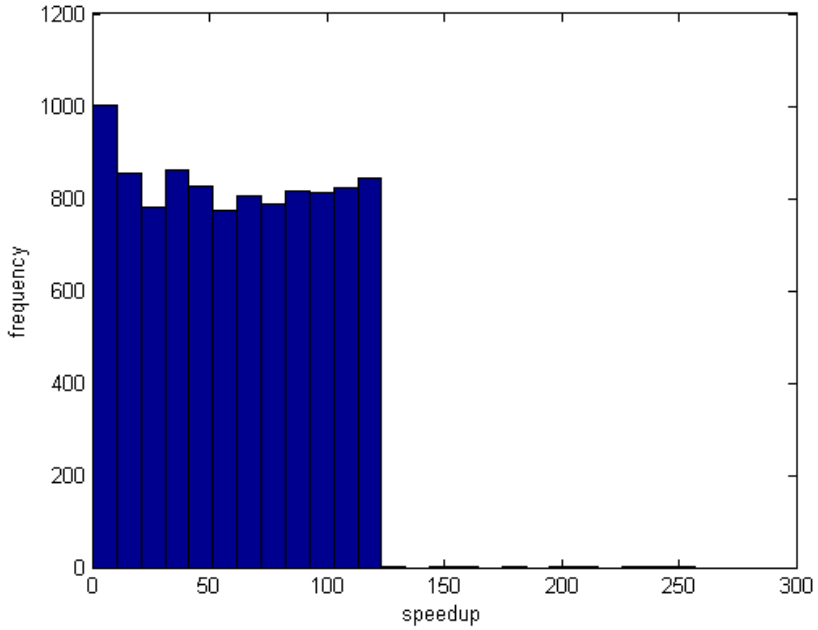

Figure 9: Distribution of the speedup of the rsync algorithm with the block-size parameter set to 8KB

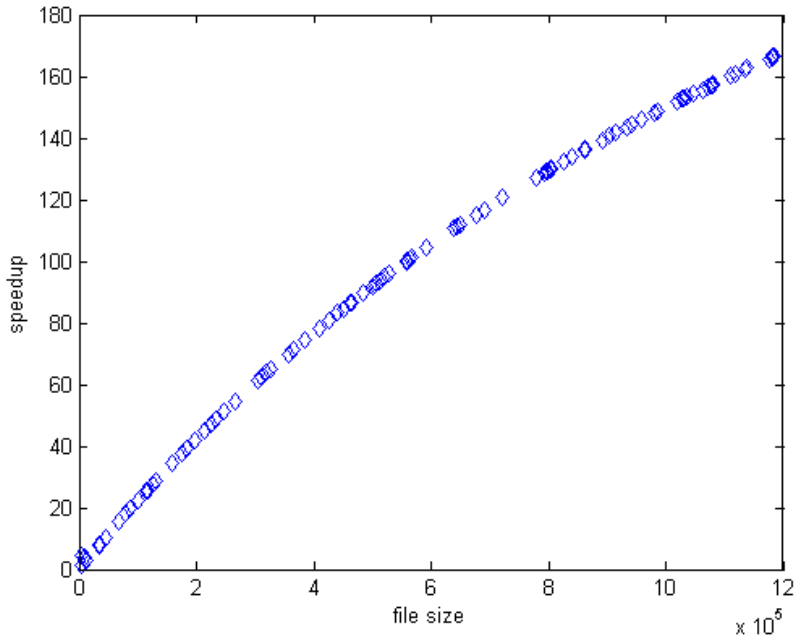

Figure 10: File-size against speedup. J. Davenport, University of Bath

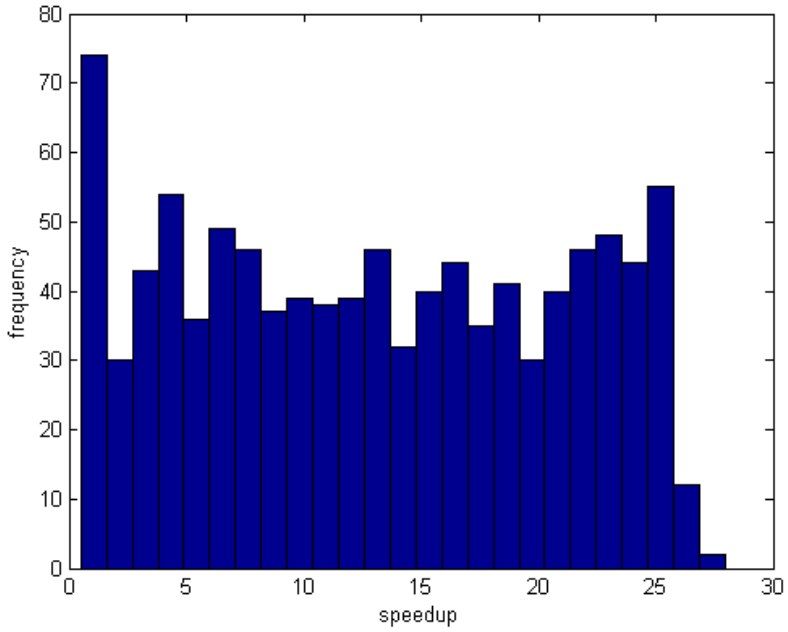

Figure 11: Distribution of the speedup of the rsync algorithm when 20 1-byte changes were made to the local file

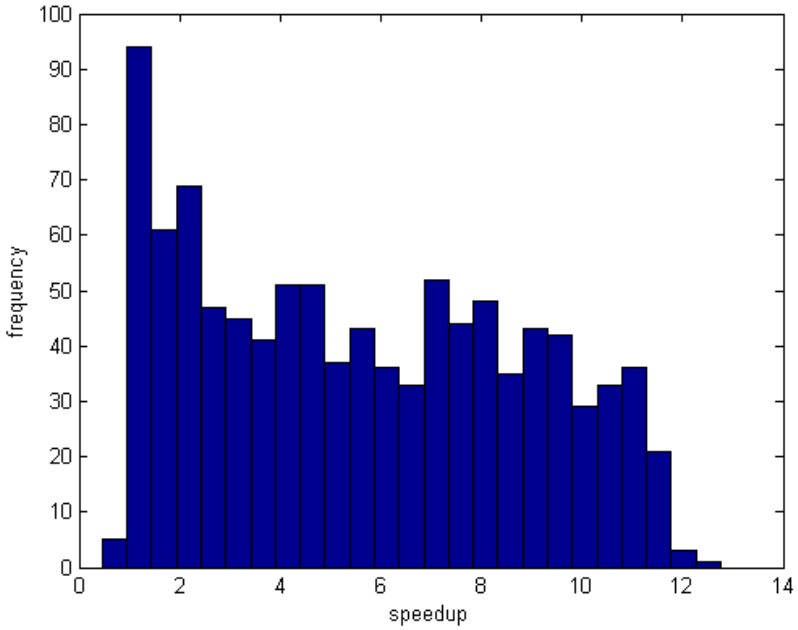

Figure 12: Distribution of the speedup of the rsync algorithm when 50 1-byte changes were made to the local file

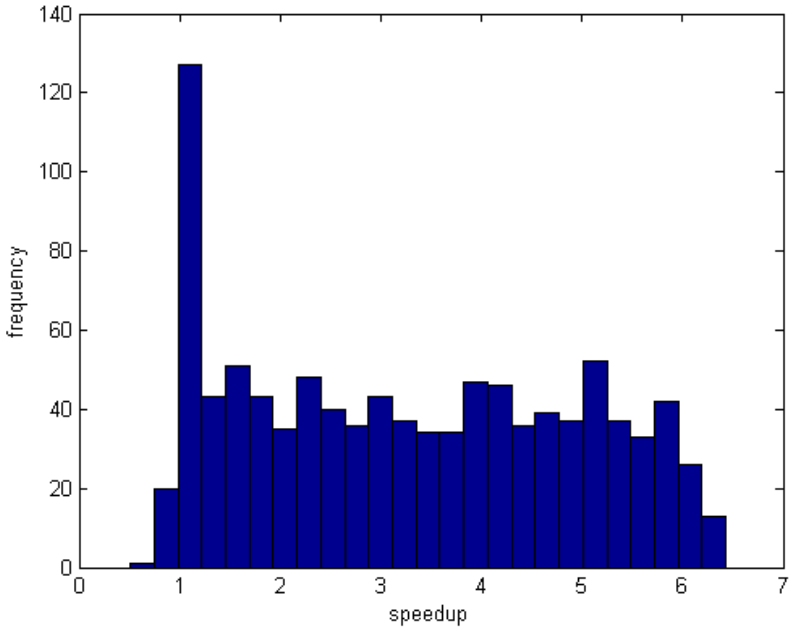

Figure 13: Distribution of the speedup of the rsync algorithm when 100 1-byte changes were made to the local file

Figure 14 illustrates the distribution of time needed to perform synchronisation when 1 change (deletion of c bytes and insertion of r bytes) is made to the local file. On the other hand Figure 15 illustrates the distribution of time needed for the rsync when 100 1-byte insertions are made to the local file. The block-size parameter was set to 2KB in the experiments that produced these plots. We observe that the time needed to synchronise in these two situations is not very different.

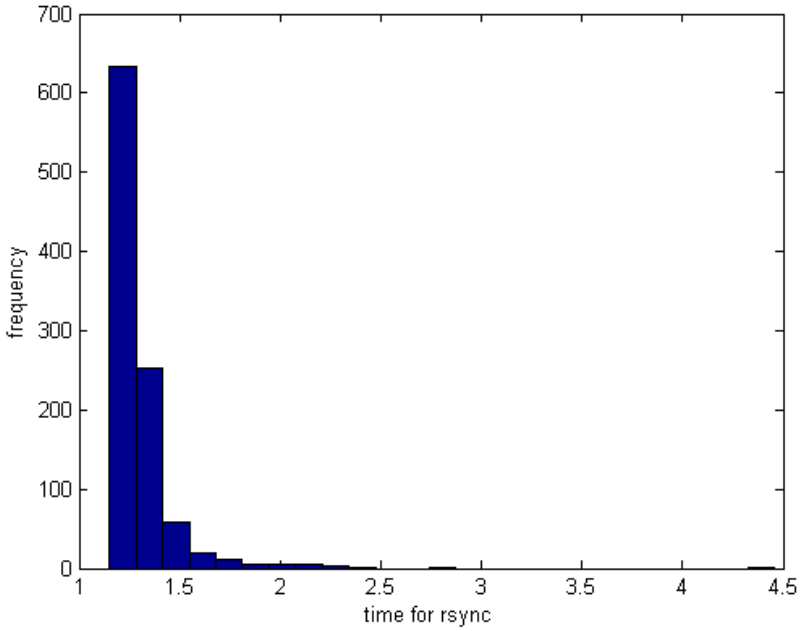

Figure 14: Distribution of time needed for rsync when 1 change was made to the local file

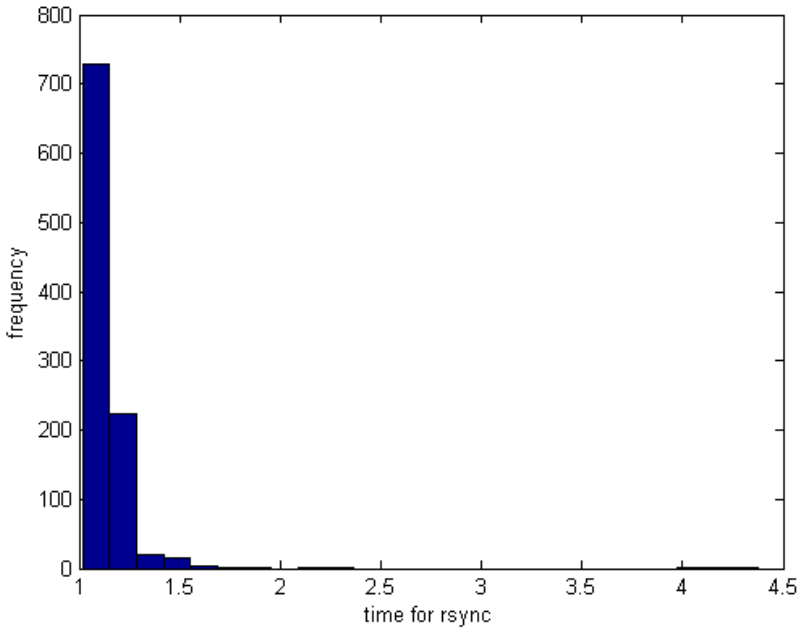

Figure 15: Distribution of time needed for rsync when 100 changes were made to the local file

4 Mathematical model

4.1 Motivation

The following example mathematical model is given in Tridgell's thesis Section 3.3 [3]. It is assumed that the two files (the local one and the one stored at the remote system) are the same except for Q sequences of bytes. Each sequence is smaller than the block-size and is separated by more than the block-size from the next sequence. Then the total number of bytes transmitted, denoted by t, is approximately

$$t(L) = (s_f + s_s)\,\frac{n}{L} + QL + s_t\left(\frac{n}{L} - Q\right) + O(1) \tag{4.4}$$

where L denotes the block-size in rsync and n the size of the changed file. Moreover s_f, s_s and s_t denote the byte size of the fast signature, the strong signature and the block match token respectively. Actually the values they take in the rsync algorithm are $s_f = 4$, $s_s = 16$ and $s_t = 4$. Figure 16 illustrates the intuition behind the model. It illustrates how the bytes are transmitted in the rsync link.

Being inspired by this mathematical model I have attempted an optimization of it.

4.2 My model

I actually left the assumption that each sequence is smaller that the block-size behind. Figure 17 and 18 give a picture of the changes to the local file allowed by the assumptions made in Tridgell's thesis [3] example model and the assumptions made in my new model respectively. Black and green colour represent the original file and the changes made to it respectively. The new model seems, indeed, more realistic. Moreover after running the rsync algorithm I noticed that the bytes that are sent to the local system A (i.e. the fast and strong signatures) are just 4 and not 20 ($s_f + s_s = 4 + 16 = 20$). One argument that explains this could be that the strong signature is not sent at all. Unfortunately I haven't investigated this further with more exepriments. In my model I replaced $s_f + s_s$ with the number 4.

I produced the following extended model:

$$t(L) = 4\frac{n}{L} + L\sum_{i=1}^{Q}\left\lceil\frac{n_i}{L}\right\rceil + 4\left(\frac{n}{L} - \sum_{i=1}^{Q}\left\lceil\frac{n_i}{L}\right\rceil\right) + O(1) \tag{4.5}$$

where n_i is how many bytes the i^{th} sequence consists of. This number is divided by L and then I get the ceiling so to obtain the number of blocks the specific sequence occupies and hence how many blocks will be sent to the file at the remote system. Then I take the sum over all the sequences.

Now, I will give an example, so to make my model clearer. Suppose the new file has size 20KB and there are 2 sequences which differ from the old file, i.e. $Q = 2$. Suppose that the first sequence has length 7KB and the second 1KB so $n_1 = 7$ and $n_2 = 1$. Let 4KB be the value of the block-size, L, and just as assumed before these differences are separated by distance 4KB or more. Figure 19 gives a picture of the new file being described here. The old file and the changes made to it are represented by the black and red colour respectively. Note that each rectangle represents a 4KB block. Then the total number of bytes transmitted is

$$t(L) = 12312 + \epsilon.$$

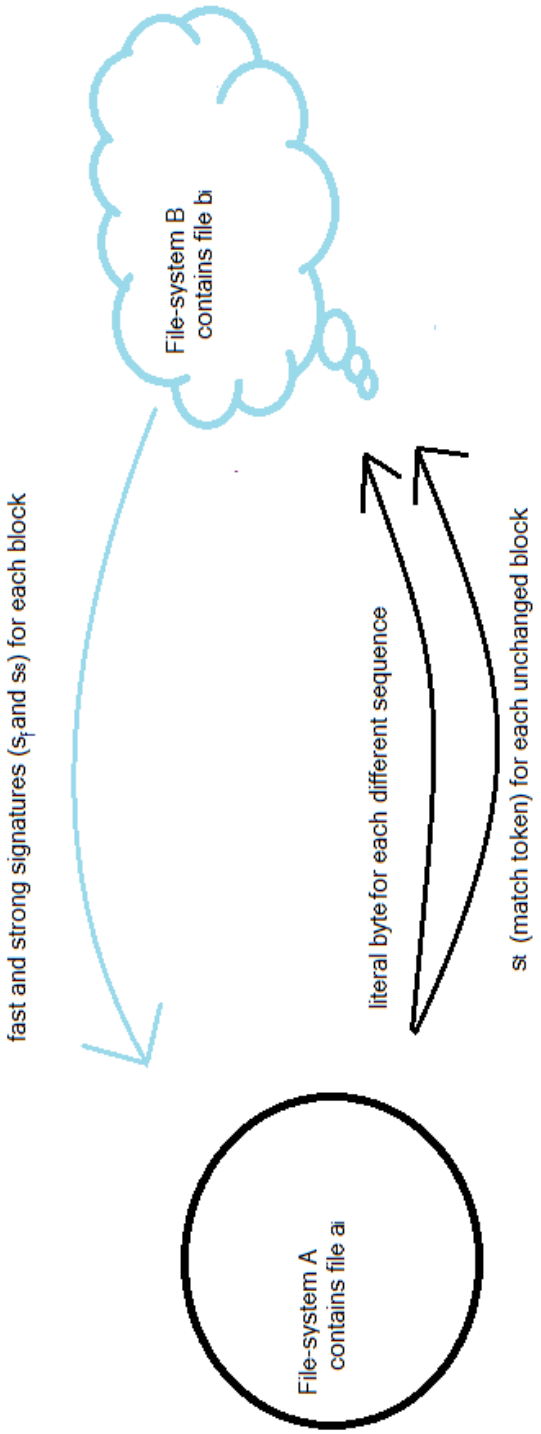

Figure 16: Bytes that are transmitted through the rsync link

Figure 17: A changed file following the assumptions of the example mathematical model given in Andrew Tridgell's thesis [3]

Figure 18: A changed file following the assumptions of my mathematical model

4.2.1 Simulations

I continue by studying the model given by the equation (4.5). First I perform simulations.

So to be consistent with my initial emulation experiments from Section 3 I had to make the four following assumptions when simulating:

1. The file-size, n bytes, follows a power-law distribution.
2. The sequences begin at random byte positions that follow a U[0,n] distibution.
3. The changes are actually deletions of c bytes and insertions of r bytes. c and r follow a power-law distribution as well. Note that $n_i = r$.
4. There is only one difference in each file.

This means that the simulated model is the following:

$$t(L) = 4\frac{n}{L} + L\left\lceil\frac{n_1}{L}\right\rceil + 4\left(\frac{n}{L} - \left\lceil\frac{n_1}{L}\right\rceil\right) + O(1) \tag{4.6}$$

Let me also add that $O(1)$ stands for the single bytes that need to be sent so to give information about the position of a difference or the number of bytes that were deleted (c) from the original file etc. These simulations were coded in Python. The codes can be found in Appendix F.

Figure 19: The changed file described in my example

To start with, the parameters of the power-law distribution the file-size follows are extracted from Figure 4 as explained in Section 2. This gives an average file-size n =546KB. The parameters of the power-law distribution that c and r follow are randomly chosen so to give values lower than the file-size value, n. The avegare size of the different sequence is $n_1 = 28$ bytes.

Varying the value of the block-size, L, in equation (4.6) and performing 10000 simulations each time I obtained the histograms in Appendix A. They illustrate the speedup distribution for different values of the block-size parameter. These histograms agree with the Figures 5, 6, 7, 8 and 9 (Section 3). The model given by the equation (4.6) is now validated as the simulations of it agree with the actual emulation experiments.

From Appendix A and the initial experiments (Figures 5, 6, 7, 8 and 9) I see that there exists a pattern in the speedup distribution as the block-size parameter varies. This has driven me into performing more simulations of the model and extracting values for the mean and confidence interval of the ratio $\frac{t}{n+n_1}$.The codes for these simulations are in Appendix F.

Table 1 was obtained by varying the blocksize parameter and performing 1000 simulations of the model given by the equation (4.6) each time.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
1024 bytes	0.00965	$(0.00935, 0.00100)$
2048 bytes	0.00755	$(0.00740, 0.00771)$
4096 bytes	0.00925	$(0.00917, 0.00932)$
6144 bytes	0.01234	$(0.01229, 0.01239)$
8192 bytes	0.01532	$(0.01528, 0.01536)$

Table 1: Mean and 95% confidence interval for transmitted bytes for several values of the block-size. The confidence interval was estimated assuming that the simulated values of t follow the normal distribution (Central Limit Theorem [6]).

It is clear from Table 1 that the optimal block-size, the one that minimizes the ratio of the number of bytes actually transmitted to the size of the whole file, is the value 2KB. Note that in Tridgell's thesis [3] speedup is defined as the ratio of the size of the source file (the local changed file) to the total number of bytes transmitted. This means that $speedup = \frac{n_1+n}{t}$. Considering this definition of the speedup, a block-size around 2KB gives the maximum speedup, as well.

Performing simulations of the model given by the equation (4.6) for the file-size

n =1MB= 1024^2 the same way as above, I deduce that in this case the optimal block-size is 3KB. More informations can be seen in Table B2 in Appendix B. This motivated me to repeat this simulation for more file-size values, observe the behaviour of the model and deduce on the optimal block-size for each file-size. All the tables for each repetition of the simulation can be found in Appendix B. Table 2 presents the optimal block-size for the file-size values investigated. There is an obvious pattern in the optimal block-size as the file-size increases. The optimal block-size increases with the file-size.

File-size	Optimal block-size
512KB	2KB
1MB	3KB
512MB	64KB
1GB	96KB

Table 2: Optimal block-size for some file-size values.

Furthermore I investigated how the optimal block-size is affected when the size of the change made to the file is varied. I obtained Table 3 after several simulations. I simulated the model given by the equation (4.6) for different change-size values and recorded the ratio of the number of bytes transmitted to the size of the whole file, each time. The tables obtained from each simulation are in Appendix D. Then I deduced on the optimal block-size for every change-size. Also, note that the average file-size was n=546KB in all the simulations here.

Change-size	Optimal block-size
1KB	2KB
2KB	2KB
4KB	4KB
8KB	4KB

Table 3: Optimal block-size for some change-size values.

Considering Table 3, it seems that the optimal block-size does depend on the size of the change as well. The longer the changes the larger the optimal block-size.

The last thing I have investigated was the way the optimal block-size is affected by the number of changes that are made; in other words how the optimal block-size depends on the sparsity of the changes. Relaxing the 4^{th} assumption of the model with (4.6) we return to the model given by equation (4.5). Note that the average file-size is 546KB. I obtained Table 4 which shows the dependence of the optimal block-size to the sparsity of the changes. More information about the procedure can be found in Appendix C.

Number of 1-byte changes	Optimal block-size
1	2048 bytes
5	1024 bytes
20	512 bytes
50	256 bytes
100	256 bytes

Table 4: Optimal block-size for different number of changes.

4.2.2 Mathematical analysis

The ceilings make it really difficult to proceed working analytically with the model given by the equation (4.5). (The ceiling function is not continuous at all so differentiation is actually impossible.) Hence I replaced $\left\lceil \frac{n_i}{L} \right\rceil$ with B_i which implied the following simplified version:

$$t(L) = 4\frac{n}{L} + L\sum_{i=1}^{Q} B_i + 4\left(\frac{n}{L} - \sum_{i=1}^{Q} B_i\right) + O(1) \tag{4.7}$$

i.e. now B_i denotes the number of blocks the i^{th} sequence occupies.

In the context of the example I gave before this means that $B_1 = 2$ and $B_2 = 1$.

Differentiating the equation (4.7) with respect to L and setting the expression equal to zero we obtain the optimal value of the block-size. This is the value that minimises the amount of bytes transferred with every synchronisation and hence minimises the network traffic. This value is $L_{\text{optimal}} = \sqrt{\frac{8n}{\sum_{i=1}^{Q} B_i}}$.

Differentiating the equation (4.7) twice and substituting the above value of L, I obtained the expression

$$t''(L_{\text{optimal}}) = \frac{16n}{\left(\sqrt{\frac{8n}{\sum_{i=1}^{Q} B_i}}\right)^3} \tag{4.8}$$

which is always positive. Hence L_{optimal}, indeed, gives a minimum of the equation (4.7) by the Second Derivative Test [9].

5 Discussion

5.1 Mathematical model evaluation

As we have seen in Section 3 and Appendix A emulations and simulations agree. This means that the experiments with real files and the real rsync validate the model given by the equation (4.6). Hence considering Section 4.2.2 I conclude that $L_{\text{optimal}} = \sqrt{\frac{8n}{B_1}}$. This gives the optimal blocksize, i.e. the blocksize at which we have minimum network traffic and hence maximum speedup, when it is provided with a file-size, n, and a change-size, B_1. When we deal with a file system instead of a single file we could use the average file-size and average change-size instead. We can even use this expression when we desire to be more general and obtain the optimal block-size when the changes in a file are more than 1. We just need to know the typical change-size, B_1, for every n bytes in a file, i.e when we have information on the sparsity of the changes the fact that this formula deals with one change size is not a limitation.

5.2 rsync algorithm speedup and time

In Section 3 we observed that tremendous speedups are achieved when rsync is called to synchronize a remote copy of a file with the local one that has one small change. This is deducted from Figures 5, 6, 7, 8 and 9. Note that in Figure 6 rsync performs optimally. On the other hand Figure 10 tells us that the larger the file the highest the speedup when a single change is made. From these two observations we can conclude that rsync alogirthm achieves a very high speedup when it is called to deal with files that have some sparse changes.

Moreover in Figures 11, 12 and 13 we can see that as the changes get less sparse the speedup value decreases but it is rarely less than 1. Even in the worst case, Figure 13, speedup values above 1 are still being observed. Moreover note that in this case if rsync was tuned to have block-size value 256 bytes (see Table 4 in Section 4.2.1) we would observe higher speedups. Hence we conclude that using the rsync algorithm causes less network traffic than transmitting the whole file. Finally in Figures 14 and 15 we notice that the time needed to perform synchronization when changes are sparse and when they are less sparse is almost the same. Thus we need not worry about latency. Keeping this in mind compined with the fact that when the changes are sparse the speedups are enormous I deduce that it is not worth the effort replacing the rsync algorithm with a "send the whole new file"command. Since this kind of command would clearly cause a speedup value of 1 no matter how many changes there are and it might need more time than rsync.

5.3 Future work

Now I would like to emphasize on that the rsync algorith is a recent invention as it was initially presented in 1999. It is outstanding how fast it operates when there are "some small" changes in a large file; as we have seen there are speedups up to 160. These high speedups can be used in systems where the users desire instant synchronization because they work on the same file simultaneously from different machines or when a tablet app needs to be updated continuously in "less than seconds" amount of time. Therefore there can be a new synchronization era if this algorithm is optimized and implemented according to the characteristics of each file-system and user's needs.

Here are some ideas for future work:

5.3.1 Mathematical model extension

An extension of the model given by the equation (4.5) is to assume that the sequences are separated by "at least one byte" rather than by "more than the block-size" number of bytes and adapt it to this. This will give a realistic model since the only assumption is that the two files are the same except of Q sequences consisting of n_i bytes each.

5.3.2 More experiments

Unfortunately I did not have enough time to complete all the experiments I planned to. I managed, though, to construct the experimental setup hence the experiments can be continued from the point I left them. My next step would be to validate the model given by the equation (4.5) by performing simulations of it and emulations with files that have more than one different sequence of bytes. I managed to code the emulation experiment for this. First a file is constructed the same way as in Section 3 then I copy it and I add single bytes at random positions of the copied version. The number of single bytes that will be added can be specified by the one experimenting. With a simple modification of the code the experimenter can add more than one byte or even delete some bytes as well. These codes can be found in Appendix E.

By validating this model the conclusion $L_{\text{optimal}} = \sqrt{\frac{8n}{B_1}}$ becomes $L_{\text{optimal}} = \sqrt{\frac{8n}{\sum_{i=1}^{Q} B_i}}$ which is clearly stronger, more useful and more applicable.

Here let me add that these experiments may produce sequences of changes that are not seperated by more than the block-size number of bytes and hence an assumption of the model would be violated. Thus there is a possibility of differences between the simulations' and emulations' results. The best scenario would be first to construct the model explained in Section 5.3.1 and then perform simulations and the experiments described above so to validate this better version of the model given by the equation (4.5).

5.3.3 Hash functions

As explained in Section 1.2 a 16 bit hash for each fast signature is computed at the signatures' matching stage of the rsync algorithm. Hence I propose a study on these hashes so to see how they affect the algorithm performance and whether the algorithm is more efficient if a different hash function is used.

5.3.4 The user model and the distribution of changes

As I have mentioned in Section 1.1 Cloud online storage is widely used. For example it is used from people in industry who work with text files, programs and data files. This kind of files experience a lot of small changes on regular basis. On the other hand some users use the Cloud just to store their movies, music and pictures. These files are changed rarely and when a single change is made the whole file is different. Hence one can construct a user model depending on which application of the rsync algorithm they want to concentrate on. A simple example of a user model is given in A.B. Downey [10].

I also suggest a study on the distribution of changes that will aim in gaining statistics about the changes for several types of files in several file-systems. The statistics will give information

about the sparsity of the changes, their style (deletion or insertion) and their size according to the characteristics of the corresponding file-system. This is a wide topic and it is directly correlated with the rsync algorithm performance. Note that the user model can be validated using the distribution of changes statistics.

Combining the model that captures the bytes that are transmitted through the rsync link, the user model that will be constructed and the distribution of changes will be very helpful in optimizing the performance of the rsync algorithm according to the user's behaviour. Moreover the frequency at which the file-change events occur will give information on the ideal frequency of the rsync events. It might be that the rsync events is better to occur after every change or that it is better to occur at fixed time intervals.

6 Conclusions

The conclusions that were drawn are the following:

1. The optimal block-size is given by $L_{\text{optimal}} = \sqrt{\frac{8n}{B_1}}$. It is characterised as optimal, because when the block-size parameter is set to L_{optimal} in the rsync algorithm the number of bytes trasmitted through the rsync link is minimum, i.e. the rsync link performance is optimal since the network traffic is minimum.

2. As file-size increases, the optimal block-size increases.

3. The larger the size of the changes the larger the optimal block-size. Moreover the optimal block-size depends on the sparsity of the changes. The sparser the changes the larger the optimal block-size value. In other words the optimal block-size needs to be decreased when the number of changes increase and the file-size is fixed.

4. The sweet spot of the block-size parameter, for this specific file-size distribution I worked with, is 2KB.

5. rsync algorithm can give the desired "weighted combination" of latency and network traffic if it is tuned according to the users needs and file system's characteristics. Hence it is an efficient and fast way to synchronize remote systems with local ones. Moreover it can be found useful in all the synchronization links that are widely used nowdays.

Appendix

A Simulations for different block-size values

I performed 10000 simulations of the model (4.6) and I obtained the Figures 20, 21, 22, 23 and 24. They illustrate the distribution of the speedup values that were obtained by performing the simulations. As it can be seen the highest mean value for the speedup is obtained when block-size=2KB.

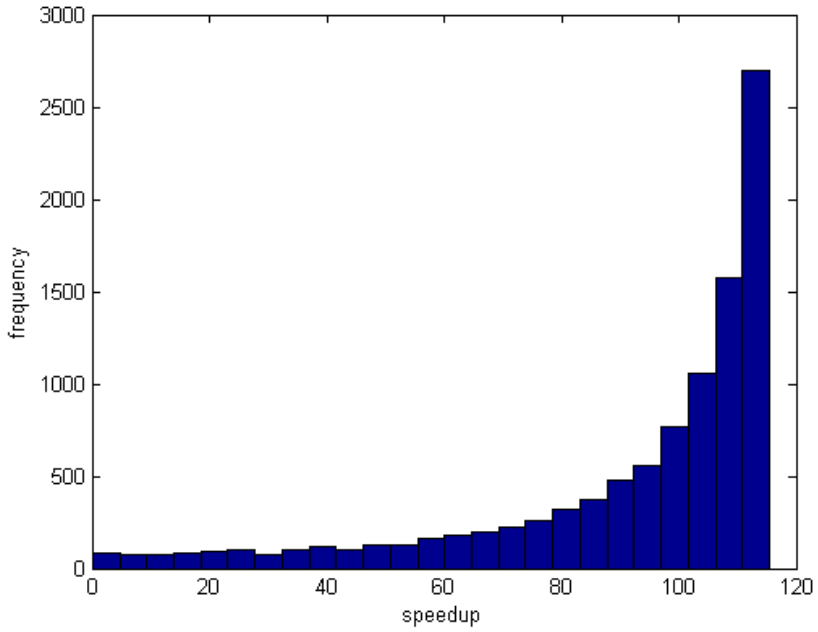

Figure 20: Distribution of the speedup of the rsync algorithm with block-size set to $1KB$

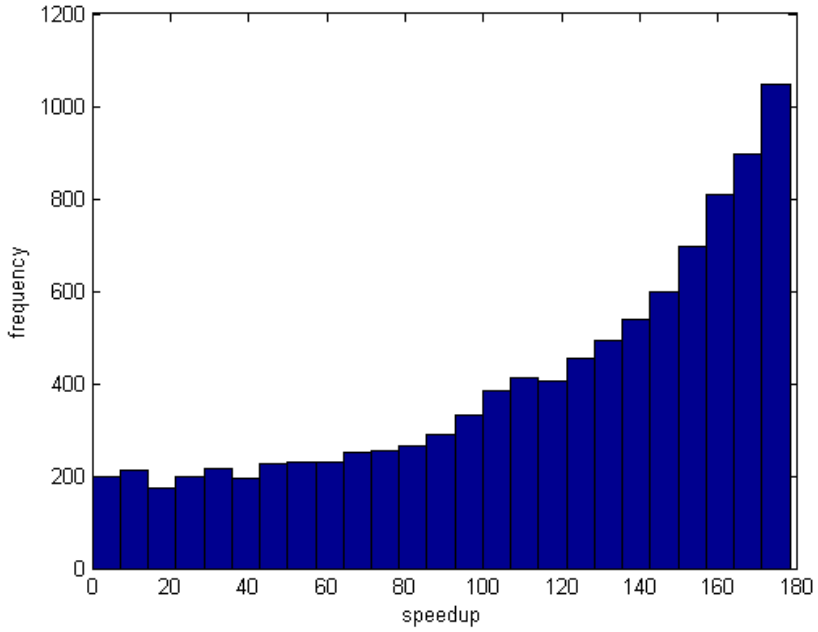

Figure 21: Distribution of the speedup of the rsync algorithm with block-size set to $2KB$

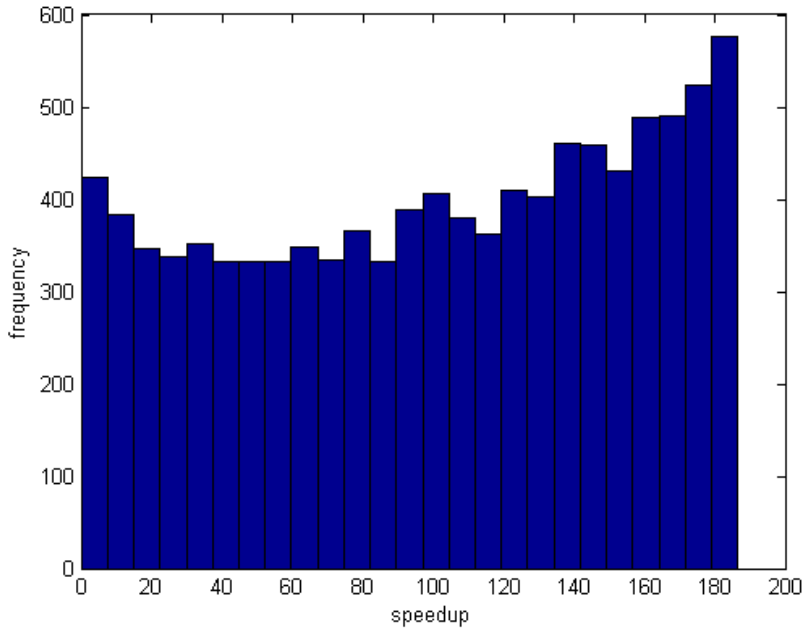

Figure 22: Distribution of the speedup of the rsync algorithm with block-size set to $4KB$

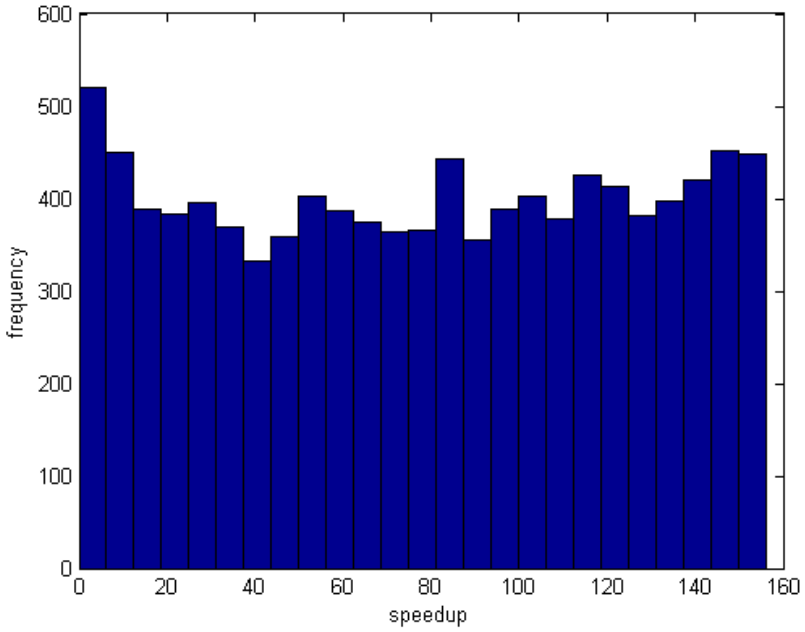

Figure 23: Distribution of the speedup of the rsync algorithm with block-size set to $6KB$

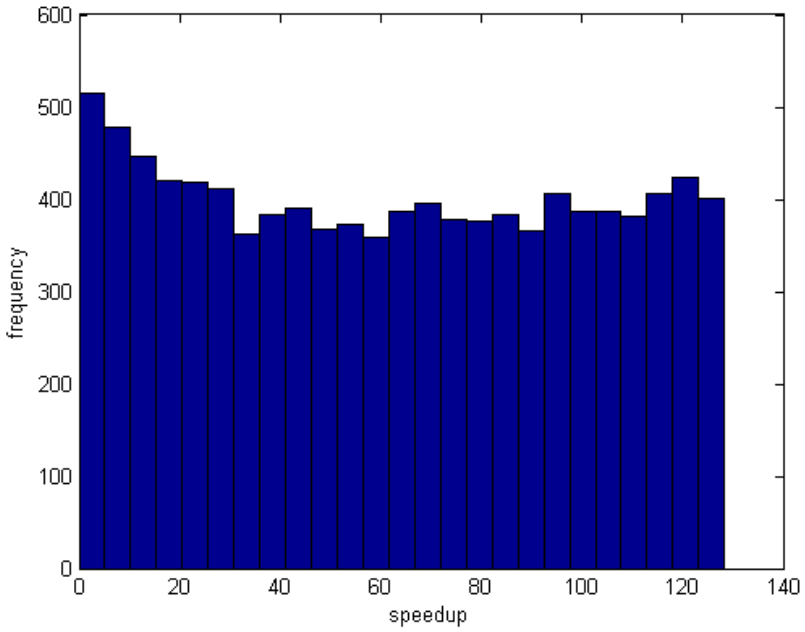

Figure 24: Distribution of the speedup of the rsync algorithm with block-size set to $8KB$

B Optimal block-size for different file-size values

The following tables illustrate the simulations carried out to decide on the optimal block-size for several file-size values. Each row of the tables was obtained by varying the block-size parameter and performing 1000 simulations. Then the mean and a confidence interval for the ratio $\frac{t}{n+n_1}$ was calculated.

As obtained in the mathematical analysis carried out in Section 4.2.2 only one value of the block-size minimizes t, the number of bytes transmitted through the rsync link. Hence, the block-size value located by these simulations of the model is indeed the desired optimal one.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
1KB	0.0097670	$(0.0097667, 0.0097674)$
2KB	0.0078137	$(0.0078133, 0.0078140)$
4KB	0.0097669	$(0.0097666, 0.0097673)$

Table B1: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for file-size n=512KB. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
2KB	0.0058602	$(0.0058600, 0.0058603)$
3KB	0.0055346	$(0.0055344, 0.0055347)$
4KB	0.0058604	$(0.0058602, 0.0058606)$

Table B2: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for file-size $n = 1MB$. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
32KB	0.0003051775	$(0.0003051772, 0.0003051778)$
64KB	0.0002441425	$(0.0002441422, 0.0002441428)$
128KB	0.0003051775	$(0.0003051772, 0.0003051778)$

Table B3: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for file-size $n = 512MB$. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
64KB	0.00018310636	$(0.00018310619, 0.00018310653)$
96KB	0.00017293374	$(0.00017293357, 0.00017293391)$
128KB	0.00018310633	$(0.00018310616, 0.00018310650)$

Table B4: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for file-size $n = 1GB$. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

C Optimal block-size for different number of changes

1000 simulations of the model (4.5), with 1 1-byte change (i.e. n_1=1byte, a single byte addition), give Table C1.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
1024 bytes	0.00971	$(0.00939, 0.01003)$
2048 bytes	0.00765	$(0.00750, 0.00780)$
3072 bytes	0.00811	$(0.00801, 0.00821)$

Table C1: Mean and 95% confidence interval for transmitted bytes for several values of the block-size. The confidence interval was estimated assuming that the simulated values of t follow the normal distribution (Central Limit Theorem [6]).

Table C1 shows that the optimal block-size is 2KB.

1000 simulations of the model (4.5)were performed. It was assumed that there are 5 1-byte changes (i.e. $n_i = 1$ for $i \in 1, 2, ..., 5$, 5 single bytes were added to the file at random positions) that are separated by more than the block-size number of bytes. Table C2 was obtained.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
512 bytes	0.0202	$(0.0196, 0.0208)$
1024 bytes	0.0170	$(0.0167, 0.0173)$
2048 bytes	0.0228	$(0.0226, 0.0229)$

Table C2: Mean and 95% confidence interval for transmitted bytes for several values of the block-size. The confidence interval was estimated assuming that the simulated values of t follow the normal distribution (Central Limit Theorem [6]).

Table C2 shows that the optimal block-size is 1KB.

1000 simulations of the model (4.5) were performed. It was assumed that there are 20 1-byte changes (i.e. $n_i = 1$ for $i \in 1, 2, ..., 20$, 20 single bytes were added to the file at random positions) that are separated by more than the block-size number of bytes. Table C3 was obtained.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
256 bytes	0.0402	$(0.0390, 0.0414)$
512 bytes	0.0336	$(0.0330, 0.0342)$
1024 bytes	0.0447	$(0.0444, 0.0450)$

Table C3: Mean and 95% confidence interval for transmitted bytes for several values of the block-size. The confidence interval was estimated assuming that the simulated values of t follow the normal distribution (Central Limit Theorem [6]).

Table C3 shows that the optimal block-size is 512 bytes.

1000 simulations of the model (4.5). It was assumed that there are 50 1-byte changes (i.e. $n_i = 1$ for $i \in 1, 2, ..., 50$, 50 single bytes were added to the file at random positions) that are separated by more than the block-size number of bytes. Table C4 was obtained.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
128 bytes	0.0734	$(0.0710, 0.0759)$
256 bytes	0.0541	$(0.0529, 0.0553)$
512 bytes	0.0610	$(0.0604, 0.0616)$

Table C4: Mean and 95% confidence interval for transmitted bytes for several values of the block-size. The confidence interval was estimated assuming that the simulated values of t follow the normal distribution (Central Limit Theorem [6]).

Table C4 shows that the optimal block-size is 256 *bytes*.

1000 simulations of the model (4.5). It was assumed that there are 100 1-byte changes (i.e. $n_i = 1$ for $i \in 1, 2, ..., 100$, 100 single bytes were added to the file at random positions) that are separated by more than the block-size number of bytes. Table C5 was obtained.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
128 bytes	0.0841	$(0.0817, 0.0865)$
256 bytes	0.0766	$(0.0753, 0.0778)$
512 bytes	0.1070	$(0.1064, 0.1076)$

Table C5: Mean and 95% confidence interval for transmitted bytes for several values of the block-size. The confidence interval was estimated assuming that the simulated values of t follow the normal distribution (Central Limit Theorem [6]).

Table C5 shows that the optimal block-size is 256 *bytes*.

As we have seen in the mathematical analysis carried out in Section 4.2.2 only one value of the block-size minimizes t, the number of bytes transmitted through the rsync link. Hence, the block-size value located by these simulations of the model is indeed the desired optimal one.

D Optimal block-size for different change-size values

The following tables illustrate the simulations carried out to decide on the optimal block-size for several change-size values. For each row of the tables the block-size parameter was varied and 1000 simulations were performed. Then the mean and a confidence interval for the ratio $\frac{t}{n+n_1}$ was calculated.

As obtained in the mathematical analysis carried out in Section 4.2.2 only one value of the block-size minimizes t, the number of bytes transmitted through the rsync link. Hence, the block-size value located by these simulations of the model is indeed the desired optimal one.

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
1KB	0.00964	$(0.00933, 0.00995)$
2KB	0.00753	$(0.00738, 0.00768)$
4KB	0.00938	$(0.00930, 0.00946)$

Table D1: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for change-size n_1=1KB. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
1KB	0.01148	$(0.01117, 0.01178)$
2KB	0.00754	$(0.00739, 0.00769)$
4KB	0.00938	$(0.00930, 0.00945)$

Table D2: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for change-size n_1 =2KB. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
2KB	0.01111	$(0.01095, 0.01126)$
4KB	0.00902	$(0.00894, 0.00909)$
6KB	0.01222	$(0.01217, 0.01227)$

Table D3: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for change-size n_1 =4KB. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

Block-size	mean value of $\frac{t}{n+n_1}$	95% CI for $\frac{t}{n+n_1}$
2KB	0.01854	$(0.01838, 0.01869)$
4KB	0.01663	$(0.01656, 0.01671)$
6KB	0.02319	$(0.02314, 0.02324)$

Table D4: Mean and 95% confidence interval for transmitted bytes for several values of the block-size and for change-size n_1 =8KB. The confidence interval was estimated assuming that the simulated values of $\frac{t}{n+n_1}$ follow the normal distribution (Central Limit Theorem [6]).

E Experiments implementation in Python 2.4.3

Create a file and name it `experiment_implementation_01.py`. Then copy and paste the following code in it.

```
#!/usr/bin/env python
# Maria Constantinou 2013

from random import random,choice
from glob import glob

def power_law(x0,x1,n,power):
    """
    generates a power law distn from a uniform distribution U[0,1] and
        then returns a value coming from this distn
    """
    z=x0**n

    return ((x1**n-z)*random()+z)**(power)

def make_changes():
    """
    chooses a random position in a file and adds a 1-byte character
    """

    home = '/u/e/mc380/project/project_final/'

    filenames=glob(home+'a_2')

    for filename in filenames: # make a 1-byte random change to each
        file
        f=open(filename,'r')
        txt=f.read()
        f.close()
        i=choice(range((len(txt)))) # pick a random byte
        newtxt=txt[:i]+choice('abcdefghijklmnopqrstuvwxyz0123456789')+txt
            [i:]
        f=open(filename,'w')
        f.write(newtxt)
        f.close()


def create_files(file_size,home,name,x0,x2,a,power):
    """
    creates a file that contains a random string and then
        produces a modified version of this file
    """
```

```
# create a file containing a random string of bytes
# this file has size file_size randomly selected from power law distn
  x=open("/dev/urandom","rb").read(int(file_size))
  file1=open(home+name,"w+")
  file1.write(x)
  file1.close()
# we make random changes to the file
# first we pick a byte position randomly
  p=file_size*(random())
# then we pick c randomly from a power law distn
  c = power_law(x0,x2,a,power)
# and r from a power law distn as well
  r = power_law(x0,x2,a,power)+10
# we create a new string of random bytes that has length r
  random_bytes=open("/dev/urandom","rb").read(int(r))
# we swap the string of random bytes with the string of random bytes
   with length c
# at the position we picked above
  y=x[:int(p)]+random_bytes+x[(int(p)+int(c)):]
  file2=open(home+name+'_2',"w+")
  file2.write(y)
  file2.close()
  return int(c),int(r)

def create_files2(file_size,home,name,x0,x2,a,power,num_changes):
  """
  creates a file that contains a random string
        and then produces a modified version of this file
  """

# create a file containing a random string of bytes
# this file has size file_size randomly selected from power law distn
  x=open("/dev/urandom","rb").read(int(file_size))
  file1=open(home+name,"w+")
  file1.write(x)
  file1.close()
  file2=open(home+name+'_2',"w+")
  file2.write(x)
  file2.close()
# add num_changes 1-byte characters at random positions of the file
# if num_changes=20 this adds 20 1-byte characters
  for i in xrange(num_changes):
      make_changes()
  c=0
  r=num_changes
```

```
    return int(c),int(r)
```

Create a file, copy and paste the following in it and name it `run_experiment_loop_00.py`.

```
#!/usr/bin/env python
# Maria Constantinou 2013

def rsync_experiment_loop(home,remote):
    """
    synchronizes files and extracts information to track performance of
        rsync
    """
    from experiment_implementation_01 import create_files, power_law,
        make_changes
    from math import exp, sqrt
    from time import clock,time
    from glob import glob
    from os import rename, wait
    from shutil import copyfile
    from subprocess import Popen,PIPE
    import re


# warm up synchronization:
# power law parameters
    n_1=((10.0**(0.25)-10.0**(-4.5))/(-13.0))+1.0
    n_2=n_1+10.0
    power=1.0/n_1
    power2=1.0/n_2
    x_1=(exp(1.0))**14.0
    x_0=exp(1.0)
    x_2=(exp(1.0))**3.0
    C_1=(n_1)/((x_1**n_1)-(x_0**n_1))
    C_2=(n_2)/((x_2**n_2)-(x_0**n_2))

# file_size is extracted form the above power law distribution
    file_size = power_law(x_0,x_1,n_1,power)
# create a file, then modify it by adding bytes at random position
    [c,r] = create_files(file_size,home,'a',x_0,x_2,n_2,power2)
# synchronize the two files
    p=Popen(['rsync','-av','--block-size=1024',home+'a_2',remote+'a'],
        shell=False,stdout=PIPE)

    wait()
# actual experiment:
    for j in xrange(10000):
      n_1=((10.0**(0.25)-10.0**(-4.5))/(-13.0))+1.0
```

```
    n_2=n_1+10.0
    power=1.0/n_1
    power2=1.0/n_2
    x_1=(exp(1.0))**14.0
    x_0=exp(1.0)
    x_2=(exp(1.0))**3.0
    C_1=(n_1)/((x_1**n_1)-(x_0**n_1))
    C_2=(n_2)/((x_2**n_2)-(x_0**n_2))

    file_size = power_law(x_0,x_1,n_1,power)
    [c,r] = create_files(file_size,home,'a',x_0,x_2,n_2,power2)

    re_speedup=re.compile(r'speedup\s+is\s+(?P<s>\d+\.\d*)')
    re_sent=re.compile(r'sent\s+(?P<sent>\d+)\s+bytes')
    re_rec=re.compile(r'received\s+(?P<sent>\d+)\s+bytes')
    start=time()

    p=Popen(['rsync','-av','--block-size=1024',home+'a_2',remote+'a'
        ],shell=False,stdout=PIPE)
    for line in p.stdout:
      m=re_speedup.search(line)
       if m: speedup=float(m.group('s'))
      m=re_sent.search(line)
       if m: sent=int(m.group('sent'))
      m=re_rec.search(line)
       if m: rec=int(m.group('sent'))

    elapsed = time() - start
    n_1=n_1-1.0
    n_2=n_2-1.0

    # print the the experimental setup characteristics and results in
        a file
    fout = open('output.txt','r');
    data=fout.read()
    fout.close()
    i=len(data)
    newdata =data[:i]+' %2.8f %2.8f%2.8f %2.8f %d %d %d %d %2.8f %2.8
        f %d %d\n' % (C_1, n_1, C_2, n_2, c,r, file_size,1024, elapsed
        , speedup, sent, rec)
    fout = open('output.txt','w');
    fout.write(newdata)
    fout.close
    wait()

home = '/u/e/mc380/project/project_final/'
```

```
remote='138.38.3.34' # address of remote computer (lyness.bath.uk)
rsync_experiment_loop(home,remote+':'+home)
```

Create a file, copy and paste the following in it and name it `run_experiment_loop_02.py`.

```
#!/usr/bin/env python
# Maria Constantinou 2013

def rsync_experiment_loop2(home,remote):
    """
    synchronizes files and extracts information to track performance
          of rsync, the number of changes (1-byte additions) that are
          made to the local file can be specified
    """
    from experiment_implementation_01 import create_files2, power_law
    from math import exp, sqrt
    from time import clock,time
    from glob import glob
    from os import rename, wait
    from shutil import copyfile
    from subprocess import Popen,PIPE
    import re

# warm up synchronization:
# power law parameters
    n_1=((10.0**(0.25)-10.0**(-4.5))/(-13.0))+1.0
    n_2=n_1+10.0
    power=1.0/n_1
    power2=1.0/n_2
    x_1=(exp(1.0))**14.0
    x_0=exp(1.0)
    x_2=(exp(1.0))**3.0
    C_1=(n_1)/((x_1**n_1)-(x_0**n_1))
    C_2=(n_2)/((x_2**n_2)-(x_0**n_2))

    num_changes=20 #number of 1-byte insertions
# file_size is extracted form the above power law distribution
    file_size = power_law(x_0,x_1,n_1,power)
# create a file, then modify it by adding bytes at random positions
    [c,r] = create_files2(file_size,home,'a',x_0,x_2,n_2,power2,
        num_changes)
# synchronize the two files
    p=Popen(['rsync','-av','--block-size=2048',home+'a_2',remote+'a'],
        shell=False,stdout=PIPE)

    wait()
```

```
# actual experiment:
  for j in xrange(1000):
    n_1=((10.0**(0.25)-10.0**(-4.5))/(-13.0))+1.0
    n_2=n_1+10.0
    power=1.0/n_1
    power2=1.0/n_2
    x_1=(exp(1.0))**14.0
    x_0=exp(1.0)
    x_2=(exp(1.0))**3.0
    C_1=(n_1)/((x_1**n_1)-(x_0**n_1))
    C_2=(n_2)/((x_2**n_2)-(x_0**n_2))

    num_changes=20
    file_size = power_law(x_0,x_1,n_1,power)
    [c,r] = create_files2(file_size,home,'a',x_0,x_2,n_2,power2,
       num_changes)

    re_speedup=re.compile(r'speedup\s+is\s+(?P<s>\d+\.\d*)')
    re_sent=re.compile(r'sent\s+(?P<sent>\d+)\s+bytes')
    re_rec=re.compile(r'received\s+(?P<sent>\d+)\s+bytes')
    start=time()

    p=Popen(['rsync','-av','--block-size=2048',home+'a_2',remote+'a'
       ],shell=False,stdout=PIPE)
    for line in p.stdout:
      m=re_speedup.search(line)
      if m: speedup=float(m.group('s'))
      m=re_sent.search(line)
      if m: sent=int(m.group('sent'))
      m=re_rec.search(line)
      if m: rec=int(m.group('sent'))

    elapsed = time() - start
    n_1=n_1-1.0
    n_2=n_2-1.0

# print the experimental setup characteristics and results in a file
    fout = open('output.txt','r');
    data=fout.read()
    fout.close()
    i=len(data)
    newdata =data[:i]+' %2.8f %2.8f %2.8f %2.8f %d %d %d %d %2.8f
       %2.8f %d %d\n' % (C_1, n_1, C_2, n_2, c,r, file_size, 2048,
       elapsed, speedup, sent, rec)
    fout = open('output.txt','w');
    fout.write(newdata)
```

```
    fout.close
    wait()

home = '/u/e/mc380/project/project_final/'
remote='138.38.3.34' # address of remote computer (lyness.bath.uk)
rsync_experiment_loop2(home, remote+':'+home)
```

Finally, compile all the above. `experiment_implementation_01.py` is used in running `run_experiment_loop_00.py` and `run_experiment_loop_02.py`.

F Mathematical model simulation in Python 2.4.3

Create a file `math_model_simulation.py` and copy and paste the following in it:

```
#!/usr/bin/env python
# Maria Constantinou 2013

from math import ceil,exp,sqrt
from random import random

def power_law(x0,x1,n,power):
    """
    generates a power law distn from a uniform distribution U[0,1]
          and then returns a value coming from this distn
    """

    x=((x1**n-x0**n)*random()+x0**n)**(power)

    return x

def math_model(L):
    """
          implementation of mathematimal model when we want to make
          one change that may consist of deletion of some bytes
          and insertion of n_i bytes
    """

# power law distribution parameters
    n_1=((10.0**(0.25)-10.0**(-4.5))/(-13.0))+1.0
    n_2=n_1+10.0
    power=1.0/(n_1)
    power2=1.0/(n_2)
    x_1=(exp(1.0))**14.0
    x_0=exp(1.0)
    x_2=(exp(1.0))**3.0
    C_1=(n_1)/((x_1**n_1)-(x_0**n_1))
    C_2=(n_2)/((x_2**n_2)-(x_0**n_2))

    n=power_law(x_0,x_1,n_1,power) # file-size
    n_i=(power_law(x_0,x_2,n_2,power2))+10.0 # change-size

    t=4.0*n/L+4.0*(n/L-ceil(n_i/L))+L*ceil(n_i/L)+10.0*random()
# equation of the model that captures the number of bytes transmitted

    speedup=(n_i+n)/t # speedup as defined in Andrew Tridgell's thesis

    return speedup, t, n, n_i, n_1, n_2, C_1, C_2
```

```
def math_model2(L,num_changes):
    """
        implementation of mathematical model when we want to make
        more than one changes that are more than the block-size
        bytes appart
    """

# power law distribution parameters
    n_1=((10.0**(0.25)-10.0**(-4.5))/(-13.0))+1.0
    n_2=n_1+10.0
    power=1.0/(n_1)
    power2=1.0/(n_2)
    x_1=(exp(1.0))**14.0
    x_0=exp(1.0)
    x_2=(exp(1.0))**3.0
    C_1=(n_1)/((x_1**n_1)-(x_0**n_1))
    C_2=(n_2)/((x_2**n_2)-(x_0**n_2))

    n=power_law(x_0,x_1,n_1,power) # file-size

    B_i=num_changes
# number of changes (one byte additions)
# if num_changes=100 this means that there are 100 1-byte additions
# that are seperated by more than the blocksize number of bytes

    t=4.0*n/L+4.0*(n/L-B_i)+L*B_i+10.0*random()
# equation of the model that captures the number of bytes transmitted

    speedup=(B_i+n)/t # speedup as defined in Andrew Tridgell's thesis

    return speedup, t, n, B_i, n_1, n_2, C_1, C_2
```

Create a file, name it `simulation_loop_00.py` and copy and paste the following in it:

```
#!/usr/bin/env python
# Maria Constantinou 2013
# mathematical model simulations are performed

from math_model_simulation import math_model
from array import array
from math import sqrt

n=10000
speedup=array('d',n*[0.0])

L=8192 # block-size
```

```
# perform n simulations of the model
for i in xrange(n):
   [speedup,t,file_size,change,n_1,n_2,C_1,C_2] =math_model(L)
# this can either be math_model or math_model2
# note that first the user needs to import the desired one
# print the simulation characteristics and results in a file
   fout = open('output_simulation.txt','r');
   data=fout.read()
   fout.close()
   i=len(data)
   newdata =data[:i]+' %2.8f %2.8f %2.8f %2.8f %d %d %d %2.8f %2.8f' %
       (C_1,n_1,C_2,n_2,change, file_size, L, speedup, t)
   fout = open('output_simulation.txt','w');
   fout.write(newdata)
   fout.close
```

Create a file, name it `simulation_loop_01.py` and copy paste the following in it:

```
#!/usr/bin/env python
# Maria Constantinou 2013
# mean and of 95% confidence interval for the ratio of bytes
# transmitted over the size of the whole file are computed

from math_model_simulation import math_model2
from array import array
from math import sqrt

n=1000
t=array('d',n*[0.0])
ratio=array('d',n*[0.0])
change=array('d',n*[0.0])
file_size=array('d',n*[0.0])
L=512.0 # block-size
num_changes=100 #number of 1-byte additions
# perfom n simulations of the model
for i in xrange(n):
   [speedup,t[i],file_size[i],change[i],n_1,n_2,C_1,C_2] =math_model2(
      L,num_changes)
# this can either be math_model or math_model2
# just remember to always import the one that is used
# and that they have different inputs

mean2=float(sum(change))/float(len(change))
mean3=float(sum(file_size))/float(len(file_size))

# ratio array
```

```
for i in range(n):
  ratio[i]=t[i]/(mean2+mean3)

# mean of the ratio
mean=sum(ratio)/len(ratio)


# standard deviation of the ratio
sum_sqrd=0.0
for count in ratio:
  sum_sqrd+=(count-mean)**2

std=sqrt(sum_sqrd/n)

# 95% confidence interval
lower_quantile=mean-1.96*std/sqrt(n)
upper_quantile=mean+1.96*std/sqrt(n)
# by central limit theorem

# print mean, standard deviation and 95% CI
print 'mean:', mean
print 'standard deviation', std
print '95% confidence interval:(',lower_quantile, ',', upper_quantile
    ,')'
```

Finally compile the above files. `math_model_simulation.py` is used in running `simulation_loop_00.py` and `simulation_loop_01.py`.

References

[1] *BT Cloud online storage.* `http://www.productsandservices.bt.com/consumerProducts/displayTopic.do?topicId=27272`

[2] R. Twisted *Cloud storage.* 2003. `http://www.twistedross.com/cloud-storage/`

[3] A. Tridgell. *Efficient Algorithms for Sorting and Synchronization, PhD Thesis.* The Australian National University, 1999.

[4] C. Keiper. *NetApp SnapMirror Block Level Incremental Backup to Tape with NetVault Backup.* Quest Software, 2012.

[5] A. Menezes, P. van Oorschot, S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[6] P. Hall. *Rates of convergence in the central limit theorem.* Australian National University, 1982.

[7] A. S. Tanenbaum, J. N. Herder, H. Bos. *File Size Distribution on UNIX Systems-Then and Now.* Vrije Universiteit, Amsterdam, The Netherlands, 2005.

[8] E. W. Weisstein. *Random Number.* From MathWorld – A Wolfram Web Resource. `http://mathworld.wolfram.com/RandomNumber.html`

[9] H. Anton, I. C. Bivens, S. Davis. *Calculus Late Transcendentals*, 9th Edition. Drexel University and Davidson College,United States, 2010.

[10] A. B. Downey. *The structural cause of file size distributions.* Wellesley College, 2001.